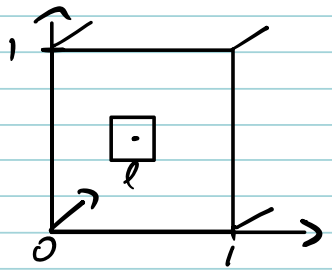
Curse of Dimensionality

Assume $x_i \in [0,1]^d$ (i.e. the d dimensional unit hypercube).
All data is drawn uniformly at random.
Let $k = 10$.

Let ℓ be the edge length of the smallest hypercube that contains all k nearest neighbors of a test point x.



$$\ell^d \approx \frac{k}{n} \Rightarrow \ell \approx \left(\frac{k}{n}\right)^{1/d}$$

ℓ^d: volume of mini cube containing the k neighbors
$\frac{k}{n}$: Total volume of hypercube $[0,1]^d$ is $1^d = 1$. $\frac{k}{n}$ is the fraction that k points take up. (because points are uniformly sampled)

If $n = 1000$ how big is ℓ?

d	2	10	100	1000
ℓ	0.1	0.63	0.955	0.9954

Almost the entire space is needed to fit 10 nearest neighbors.

This means nearest neighbors are not similar, violating the k-NN assumption!

How many points would we need for ℓ to be small?
Fix $\ell = 0.1$

$$\ell^d = \frac{k}{n} \Rightarrow n = k\left(\frac{1}{\ell}\right)^d = k\,10^d$$ ← grows exponentially with d!

Rescue to the curse:

Data can have structure:

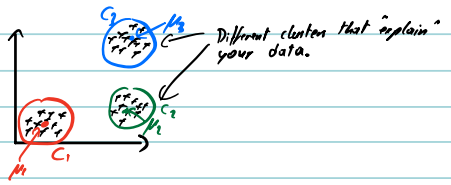
- Data can lie on intrinsically low dimensional subspaces or sub-manifolds.
- Data can be clustered (very non-uniform).

Unsupervised Learning

Given: Unlabeled data $\mathcal{D} = \{x_1, \dots, x_n\} \subseteq \mathbb{R}^d$ ← No labels y_i

Unsupervised learning attempts to find (hidden) structure in your data.

Clustering: K-Means



Assumptions: - We know there are k clusters
- Similar points share the same cluster.

Let data indices be $\{1, \dots, n\}$. Let $C_\ell \subseteq \{1, \dots, n\}$ be all points that fall into the ℓ-th cluster.

Every point falls exactly into one cluster: $C_1 \cup C_2 \cup \dots \cup C_k = \{1, \dots, n\}$

(Clusters are mutually exclusive.) $C_\ell \cap C_{\ell'} = \emptyset \quad \forall \ell \neq \ell'$

Cluster center: $\vec{\mu}_\ell = \frac{1}{|C_\ell|} \sum_{i \in C_\ell} \vec{x}_i$

Loss function: $\mathcal{L}(C_1, \dots, C_k) = \frac{1}{n} \sum_{\ell=1}^{k} \sum_{i \in C_\ell} \|\vec{x}_i - \vec{\mu}_\ell\|_2^2$ ← Average distance from each point to its cluster center.

↳ If you were to represent each $\vec{x}_i$ with its corresponding $\vec{\mu}_\ell$ instead, this would be the avg. squared error.

① Minimize $\mathcal{L}$ if $C_1, \dots, C_k$ are fixed: (Assign $\mu_1, \dots, \mu_k$)

$\vec{\mu}_\ell = \underset{\vec{\mu}_\ell}{\operatorname{argmin}} \mathcal{L}(C_1 \dots C_k) \Rightarrow \vec{\mu}_\ell = \frac{1}{|C_\ell|} \sum_{i \in C_\ell} \vec{x}_i$ ← mean of all $x_i \in C_\ell$

For each μ_ℓ, the contribution to the loss is $\sum_{i \in C_\ell} (x_i - \mu_\ell)^2$ ← Parabola

② Minimize $\mathcal{L}$ if $\mu_1, \dots, \mu_k$ are fixed: (Assign $C_1, \dots, C_k$)

$C_\ell = \{i \in \{1, \dots, n\} : \ell = \underset{1 \leq \ell' \leq k}{\operatorname{argmin}} \|x_i - \mu_{\ell'}\|^2\}$

All points that are closest to μ_ℓ.

For each x_i you can contribute $(\vec{x}_i - \vec{\mu}_1)^2$ or $(\vec{x}_i - \vec{\mu}_2)^2$ or ... or $(\vec{x}_i - \vec{\mu}_k)^2$.

For each x_i pick the smallest one to minimize $\mathcal{L}$.

K-means algorithm: (aka Lloyd's algorithm)

1) Initialize $\vec{\mu}_1, \dots, \vec{\mu}_k$ somehow (e.g. random point, or heuristic)
2) Assign $C_1, \dots, C_k$ using ②
3) Assign $\vec{\mu}_1, \dots, \vec{\mu}_k$ using ①

Repeat until convergence

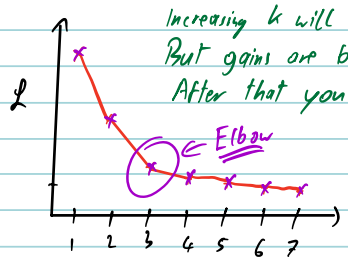
Easy to show: $\mathcal{L}$ can never increase.

Initialization: - Can be unlucky. Perform multiple runs, pick solution with lowest $\mathcal{L}$.

- k-means++ (Arthur et al. 2007): - pick μ_1 randomly from $\mathcal{D}$
  - For $\ell : 2$ to k:
    - $\forall x_i \in \mathcal{D}$ define $d_i^2 = \min_{\ell'} (x_i - \mu_{\ell'})^2$
    - pick μ_ℓ randomly from $\mathcal{D}$ proportional to d_i^2. ↳ Avoids cluster centers that are very close

How to find k if it is unknown?

Run k-means for $k = 1, 2, \ldots$ monitor the loss $\mathcal{L}$
(For each k you have to average over multiple runs, due to the randomness of the initialization.)



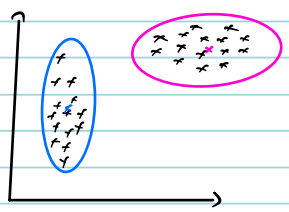
Increasing k will always decrease the best achievable $\mathcal{L}$.
But gains are bigger until you discovered the "true" k.
After that you will only split cluster into two.

Strength:
- Easy to implement
- fast for small k
- easily parallelizable

Limitations:
- sensitive to outliers
- slow for large k
- assumes all clusters are spherical
- makes only hard cluster assignments

Generalization: Gaussian Mixture Model (GMM)

Each cluster becomes a Gaussian $\mathcal{N}(\vec{\mu}_\ell, \Sigma_\ell)$



Almost the same as k-means, but each point has a distribution over clusters $\gamma_{\ell i} = p(C_\ell | \vec{x}_i)$.

$$p(C_\ell | \vec{x}_i) = \frac{p(\vec{x}_i | C_\ell) p(C_\ell)}{\sum_{\ell'} p(\vec{x}_i | C_{\ell'}) p(C_{\ell'})} \quad \text{(Bayes Rule)}$$

$$\vec{\mu}_\ell \leftarrow \frac{1}{\sum_j \gamma_{\ell j}} \sum_{i=1}^{n} \gamma_{\ell i} \vec{x}_i \leftarrow \text{weighted average / expected center of cluster } \ell$$

$$\vec{\Sigma}_\ell \leftarrow \frac{1}{\sum_j \gamma_{\ell i}} \sum_{i=1}^{n} \gamma_{\ell i} (\vec{x}_i - \vec{\mu}_\ell)(\vec{x}_i - \vec{\mu}_\ell)^T$$

$$\pi_\ell \leftarrow \frac{\sum \gamma_{\ell i}}{n} \leftarrow \pi_\ell = p(C_\ell)$$

M-step
Maximization

$$\gamma_{\ell i} \leftarrow \frac{p(x_i; \mu_\ell, \Sigma_\ell)\, \pi_\ell}{\sum_{\ell'} p(x_i; \mu_{\ell'}, \Sigma_{\ell'})\, \pi_{\ell'}}$$

E-Step
Estimation

Strengths:
- more flexible than k-means
- probabilities tell you how well a sample fits into cluster

Weaknesses:
- slower than k-means
- can suffer from singularities (single point clusters)

(If $\Sigma_\ell = I\sigma^2$ GMM becomes k-means as $\sigma^2 \to 0$.)

$\Sigma_\ell = I\sigma^2$: spherical
$\sigma^2 \to 0$: hard assignments